z/Architecture

The Enhanced-Sort Facility for z/Architecture

SA22-1082-00

Note:

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v. A revision bar indicates an addition in this version.

September, 2020

The information contained herein should not be construed as implying any intention by IBM to provide these facilities on models other than those described herein.

This document is provided for use in conjunction with other relevant IBM documents, and IBM makes no warranty, express or implied, relative to its completeness or accuracy. The information in this document is subject to change without notice.

Notices

References in this document to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product is not intended to state or imply that only IBM's product may be used. Any functionally equivalent product may be used; however, this does not constitute a waiver in any way of IBM's intellectual property rights.

The z/Architecture definitions assign meanings to various fields and bit positions in control formats such as instructions, program-status words, channel-controls words, table entries (region-table entry, page-table entry, linkage-table entry, etc.), control registers, and control blocks (e.g., the service-call-control block). Other fields and bit positions in those control formats are specified to be reserved, ignored, or checked for zeros.

In order for the machine to operate as specified in the *z/Architecture Principles of Operation* and other documents that define the attachment interface, the program should place zeros in the unassigned fields and bit positions and should not depend on the contents of these fields and bit positions to contain a specific value. IBM may offer functional extensions or performance assists that assign meanings to these fields and bit positions.

IBM may have patents or pending patent applications covering subject matter described herein. Furnishing this document does not constitute or imply a grant of any license under any patents, patent applications, trademarks, copyrights, or other rights of IBM or of any third party, or any right to refer to IBM in any advertising or other promotional or marketing activities. IBM assumes no responsibility for any infringement of patents or other rights that may result from the use of this document or from the manufacture, use, lease, or sale of apparatus described herein.

Trademarks

The following terms are trademarks or registered trademarks of the International Business Machines Corporation in the United States or other countries:

IBM
IBM Z
IBM Z family
IBM z Systems
IBM z15
z/Architecture
z Systems
z15

Related Documents

This document extends the *z/Architecture Principles of Operation* (SA22-7832-12), September, 2019.

Preface

This document describes the enhanced-sort facility. The facility provides a means to sort multiple lists of unsorted-input data into one or more lists of sorted-output data. The facility also provides a means to merge multiple lists of sorted-input data into a single list of sorted-output data. The facility includes the SORT LISTS instruction.

The description of the enhanced-sort facility is in the form of updates to Reference [1] (listed below) with which the reader is assumed to be familiar. Chapter and section headings are provided to aid in locating the affected material; material that is not updated is omitted, as indicated by a vertical ellipsis (⋮). A change bar in the left margin identifies all changed material in [1]. Cross references to unchanged material in [1] are left unresolved.

Other Publications

1. *IBM z/Architecture Principles of Operation* (SA22-7832-12), dated September, 2019.

⋮

Chapter 4. Control

⋮

Facility Indications

⋮

Figure 4-1 shows the meanings of the assigned facility bits.

⋮

Bit	Meaning When Bit Is One
⋮	⋮
150	The enhanced-sort facility is installed in the z/Architecture architectural mode.
⋮	⋮

Figure 4-1. Assigned Facility Bits

⋮

Chapter 5. Program Execution

⋮

Interruptible Instructions

Point of Interruption

For most instructions, the entire execution of an instruction is one operation. An interruption is permitted between operations; that is, an interruption can occur after the performance of one operation and before the start of a subsequent operation.

⋮

Condition-Code Alternative to Interruptibility

The following instructions are not interruptible instructions but instead may be completed after performing a CPU-determined subportion of the processing specified by the parameters of the instructions:

- CHECKSUM
- CIPHER MESSAGE
- CIPHER MESSAGE WITH CIPHER FEEDBACK
- CIPHER MESSAGE WITH CHAINING
- CIPHER MESSAGE WITH COUNTER
- CIPHER MESSAGE WITH OUTPUT FEEDBACK
- COMPARE LOGICAL LONG EXTENDED
- COMPARE LOGICAL LONG UNICODE
- COMPARE LOGICAL STRING
- COMPUTE INTERMEDIATE MESSAGE DIGEST
- COMPUTE LAST MESSAGE DIGEST
- COMPUTE MESSAGE AUTHENTICATION CODE
- CONVERT UTF-16 TO UTF-32
- CONVERT UTF-16 TO UTF-8
- CONVERT UTF-32 TO UTF-16
- CONVERT UTF-32 TO UTF-8
- CONVERT UTF-8 TO UTF-16
- CONVERT UTF-8 TO UTF-32
- MOVE LONG EXTENDED
- MOVE LONG UNICODE
- MOVE STRING
- PERFORM CRYPTOGRAPHIC COMPUTATION
- PERFORM PSEUDORANDOM NUMBER OPERATION (generate operation only)
- SEARCH STRING
- SEARCH STRING UNICODE
- SORT LISTS
- TRANSLATE AND TEST EXTENDED
- TRANSLATE AND TEST REVERSE EXTENDED
- TRANSLATE EXTENDED
- TRANSLATE ONE TO ONE
- TRANSLATE ONE TO TWO
- TRANSLATE TWO TO ONE
- TRANSLATE TWO TO TWO

When any of the above instructions is completed after performing only a CPU-determined amount of processing instead of all specified processing, the instruction sets condition code 3. On such completion, the instruction address in the PSW designates the next sequential instruction, and the operand parameters of the instruction have been adjusted so that the processing of the instruction can be resumed simply by branching back to the instruction to execute it again. When the instruction has performed all specified processing, it sets a condition code other than 3.

The points at which any of the above instructions may set condition code 3 are comparable to the points of interruption of an interruptible instruction, and the amount of processing between adjacent points is comparable to a unit of operation of an interruptible instruction. However, since the instruction is not interruptible, each execution is considered the execution of one unit of operation.

Completion with the setting of condition code 3 permits interruptions to occur. Depending on the model and the instruction, condition code 3 may or may not be set when there is not a need for an interruption.

When a storage-alteration PER event is recognized, fewer than 4K additional bytes are stored before the instruction is completed (with condition code 3, if no other condition takes precedence), and the event is indicated by an interruption. When a zero-address-detection PER event is recognized, the instruction is completed (with condition code 3, if no other condi-

tion takes precedence), and the event is indicated by an interruption.

⋮

Transactional-Execution Facility Instructions

⋮

Restricted Instructions

When the CPU is in the transactional-execution mode, attempted execution of certain instructions is restricted and causes the transaction to be aborted.

When issued in the constrained transactional-execution mode, attempted execution of restricted instructions may also result in a transaction-constraint program interruption, or may result in execution proceeding as if the transaction was not constrained. See “Constrained Transaction” on page 5-131 for further details.

Restricted instructions include all instructions that are not defined in Chapters 7-9 and 18-24 of this document and the following nonprivileged instructions.

- COMPARE AND SWAP AND STORE
- PERFORM LOCKED OPERATION
- PERFORM PROCESSOR ASSIST
- PREFETCH DATA (RELATIVE LONG), when the code in the M_1 field is 6 or 7
- STORE CHARACTERS UNDER MASK HIGH, when the M_3 field is zero and the code in the R_1 field is 6 or 7
- STORE FACILITY LIST EXTENDED
- SUPERVISOR CALL

Under the conditions listed below, the following instructions are restricted:

- BRANCH AND LINK (BALR), BRANCH AND SAVE (BASR), and BRANCH AND SAVE AND SET MODE, when the R_2 field of the instruction is nonzero and branch tracing is enabled
- BRANCH AND SAVE AND SET MODE and BRANCH AND SET MODE, when the R_2 field is nonzero and mode tracing is enabled; SET ADDRESSING MODE, when mode tracing is enabled
- MONITOR CALL, when a monitor-event condition is recognized

When the CPU is in the transactional-execution mode, it is model dependent whether the following instructions are restricted:

- CIPHER MESSAGE
- CIPHER MESSAGE WITH CIPHER FEEDBACK
- CIPHER MESSAGE WITH CHAINING
- CIPHER MESSAGE WITH COUNTER
- CIPHER MESSAGE WITH OUTPUT FEEDBACK
- COMPRESSION CALL
- COMPUTE INTERMEDIATE MESSAGE DIGEST
- COMPUTE LAST MESSAGE DIGEST
- COMPUTE MESSAGE AUTHENTICATION CODE
- CONVERT UNICODE-16 TO UNICODE-32
- CONVERT UNICODE-16 TO UNICODE-8
- CONVERT UNICODE-32 TO UNICODE-16
- CONVERT UNICODE-32 TO UNICODE-8
- CONVERT UNICODE-8 TO UNICODE-16
- CONVERT UNICODE-8 TO UNICODE-32
- PERFORM CRYPTOGRAPHIC COMPUTATION
- PERFORM PSEUDORANDOM NUMBER OPERATION
- SORT LISTS

⋮

Constrained Transaction

In the absence of repeated interruptions or conflicts with other CPUs or the channel subsystem, a constrained transaction will eventually complete, thus an abort-handler routine is not required. A constrained transaction is initiated by the TRANSACTION BEGIN (TBEGINC) instruction when the transaction nesting depth is initially zero. A constrained transaction is subject to the following constraints:

1. The transaction executes no more than 32 instructions, not including the TRANSACTION BEGIN (TBEGINC) and TRANSACTION END instructions.

2. All instructions in the transaction must be within 256 contiguous bytes of storage, including the TRANSACTION BEGIN (TBEGINC) and any TRANSACTION END instructions.

3. In addition to all instructions listed in the section "Restricted Instructions" on page 5-2, the following restrictions apply to a constrained transaction.

   a. Instructions are limited to those defined in Chapter 7, "General Instructions."

   b. Branching instructions are limited to the following:

      - BRANCH RELATIVE ON CONDITION in which the M_1 field is nonzero and the RI_2 field contains a positive value
      - BRANCH RELATIVE ON CONDITION LONG in which the M_1 field is nonzero, and the RI_2 field contains a positive value that does not cause address wraparound.
      - COMPARE AND BRANCH RELATIVE, COMPARE IMMEDIATE AND BRANCH RELATIVE, COMPARE LOGICAL AND BRANCH RELATIVE, and COMPARE LOGICAL IMMEDIATE AND BRANCH RELATIVE in which the M_3 field is nonzero and the RI_4 field contains a positive value

      (that is, only forward branches with nonzero branch masks)

   c. Except for TRANSACTION END and instructions which cause a specific-operand serialization, instructions which cause a serialization function are restricted.

   d. All SS- and SSE-format instructions are restricted.

   e. All of the following general instructions are restricted.

      - BRANCH PREDICTION PRELOAD
      - BRANCH PREDICTION RELATIVE PRELOAD
      - CHECKSUM
      - CIPHER MESSAGE
      - CIPHER MESSAGE WITH CIPHER FEEDBACK
      - CIPHER MESSAGE WITH CHAINING
      - CIPHER MESSAGE WITH COUNTER
      - CIPHER MESSAGE WITH OUTPUT FEEDBACK
      - COMPARE AND FORM CODEWORD
      - COMPARE LOGICAL LONG, COMPARE LOGICAL LONG EXTENDED, and COMPARE LOGICAL LONG UNICODE
      - COMPARE LOGICAL STRING
      - COMPARE UNTIL SUBSTRING EQUAL
      - COMPRESSION CALL
      - COMPUTE INTERMEDIATE MESSAGE DIGEST
      - COMPUTE LAST MESSAGE DIGEST
      - COMPUTE MESSAGE AUTHENTICATION CODE
      - ⋮
      - TRANSLATE EXTENDED
      - TRANSLATE ONE TO ONE, TRANSLATE ONE TO TWO, TRANSLATE TWO TO ONE, and TRANSLATE TWO TO TWO
      - UPDATE TREE

⋮

Sequence of Storage References

⋮

Storage-Operand Consistency

Single-Access References

A fetch reference is said to be a single-access reference if the value is fetched in a single access to each byte of the data field. In the case of overlapping operands, the location may be accessed once for each operand. A store-type reference is said to be a single-access reference if a single store access occurs to each byte location within the data field. An update reference is said to be single access if both the fetch and store accesses are each single access.

Except for the accesses associated with multiple-access references and the stores associated with storage change and restoration for DAT-associated access exceptions, all storage-operand references are single-access references.

Multiple-Access References

In some cases, multiple accesses may be made to all or some of the bytes of a storage operand. The following cases may involve multiple-access references:

1. The storage operands of the following instructions:
   - CHECKSUM
   - ⋮
   - UPDATE TREE
2. The storage operands of MOVE LONG and MOVE LONG EXTENDED, when the padding character is not B1 hex.
3. The stores into that portion of the first operand of MOVE LONG, MOVE LONG EXTENDED, or MOVE LONG UNICODE which is filled with padding bytes.
4. The storage operands of the decimal instructions.
5. The main-storage operands of PAGE IN and PAGE OUT.
6. The storage operands of the I/O instructions.
7. The stores into a trace entry.
8. The stores associated with the stop-and-store-status and store-status-at-address SIGNAL PROCESSOR orders.
9. The trap control block and trap save area used by TRAP.
10. The operands, dictionaries, and symbol-translation table of COMPRESSION CALL.
11. The operands, parameter block, continuation-record-recall buffer, and all input lists in the active state of SORT LISTS.

When a storage-operand store reference to a location is not a single-access reference, the value placed at a byte location is not necessarily the same for each store access; thus, intermediate results in a single-byte location may be observed by other CPUs and by channel programs.

⋮

Chapter 6. Interruptions

⋮

Data-Exception Code (DXC)

When a data exception causes a program interruption, a data-exception code (DXC) is stored at location 147, and zeros are stored at locations 144-146. The DXC distinguishes between the various types of data-exception conditions. When the AFP-register (additional floating-point register) control bit, bit 45 of control register 0, is one, the DXC is also placed in the DXC field of the floating-point-control (FPC) register. The DXC field in the FPC register remains unchanged when any other program exception is reported. The DXC is an 8-bit code indicating the specific cause of a data exception. The data exceptions and data-exception codes are shown in Figure 6-1 and Figure 6-2 on page 6-2.

Priority of Program Interruptions for Data Exceptions

DXC 2 and 3 are mutually exclusive and are of higher priority than any other DXC. Thus, for example, DXC 2 (BFP instruction) takes precedence over any IEEE exception; and DXC 3 (DFP instruction) takes precedence over any IEEE exception or simulated IEEE exception. As another example, if the conditions for both DXC 3 (DFP instruction) and DXC 1 (AFP register) exist, DXC 3 is reported.

When both a specification exception and an AFP-register data exception or a vector-instruction data exception apply, it is unpredictable which one is reported.

DXC (Hex)	Data Exception
00	General operand
01	AFP register
02	BFP instruction
03	DFP instruction
04	Quantum Exception
07	Simulated Quantum Exception
08	IEEE inexact and truncated
0B	Simulated IEEE inexact
0C	IEEE inexact and incremented
10	IEEE underflow, exact
13	Simulated IEEE underflow, exact
18	IEEE underflow, inexact and truncated
1B	Simulated IEEE underflow, inexact
1C	IEEE underflow, inexact and incremented
20	IEEE overflow, exact
23	Simulated IEEE overflow, exact
28	IEEE overflow, inexact and truncated
2B	Simulated IEEE overflow, inexact
2C	IEEE overflow, inexact and incremented
40	IEEE division by zero
43	Simulated IEEE division by zero
80	IEEE invalid operation
83	Simulated IEEE invalid operation
FE	Vector instruction
FF	Compare-and-trap instruction

Figure 6-1. Data-exception codes (DXC)

Exception	Applicable Instruction Types	CR0.45	FPC Mask	FPC Flag	DXC (Binary)	Instruction Action	DXC Placed in Real Loc. 147	DXC Placed in FPC Byte 2
General operand	Various[1]	0	none	none	0000 0000	Suppress or Terminate	Yes	No
		1					Yes	Yes
AFP register	FPS & HFP	0*	none	none	0000 0001	Suppress	Yes	No
BFP instruction	BFP	0*	none	none	0000 0010	Suppress	Yes	No
DFP instruction	DFP	0*	none	none	0000 0011	Suppress	Yes	No
IEEE invalid operation	ICMP	1*	0.0	1.0	1000 0000	Suppress[2]	Yes	Yes
IEEE division by zero	ICMP	1*	0.1	1.1	0100 0000	Suppress[2]	Yes	Yes
IEEE overflow	ICMP	1*	0.2	1.2	0010 xy00	Complete	Yes	Yes
IEEE underflow	ICMP	1*	0.3	1.3	0001 xy00	Complete	Yes	Yes
IEEE inexact	ICMP	1*	0.4	1.4	0000 1y00	Complete	Yes	Yes
Quantum Exception	ICMP	1*	0.5	1.5	0000 0100	Complete	Yes	Yes
Simulated IEEE invalid operation	IXS	1*	0.0	1.0	1000 0011	Complete	Yes	Yes
Simulated IEEE division by zero	IXS	1*	0.1	1.1	0100 0011	Complete	Yes	Yes
Simulated IEEE overflow	IXS	1*	0.2	1.2	0010 w011	Complete	Yes	Yes
Simulated IEEE underflow	IXS	1*	0.3	1.3	0001 w011	Complete	Yes	Yes
Simulated IEEE inexact	IXS	1*	0.4	1.4	0000 1011	Complete	Yes	Yes
Simulated Quantum Exception	IXS	1*	0.5	1.5	0000 0111	Complete	Yes	Yes
Vector instruction	VEC	0†	none	none	1111 1110	Suppress	Yes	Unp
		1‡					Yes	Yes
Compare-and-trap instruction	CT & LT	0	none	none	1111 1111	Complete	Yes	No
		1					Yes	Yes

Explanation:

[1] General-operand data exception applies to the decimal instructions (Chapter 8), the general instructions COMPRESSION CALL, CONVERT TO BINARY and PERFORM PSEUDORANDOM NUMBER OPERATION (Chapter 7), the DFP instructions CONVERT FROM PACKED, CONVERT FROM SIGNED PACKED, CONVERT FROM UNSIGNED PACKED, and CONVERT FROM ZONED (Chapter 20) , and the specialized-function-assist instruction SORT LISTS (Chapter 26).

[2] When the FPC mask bit corresponding to the exception condition is one, the DXC is stored in the FPC register, even though the resulting data-exception program interruption is considered to be suppressing.

⋮ ⋮

VEC Vector instructions (Chapters 21, 22, 23, and 24).

Figure 6-2. Data Exceptions

⋮

Data Exception

The data exceptions are shown in Figure 6-2 on page 6-2. A mask bit may or may not control whether an interruption occurs, as noted for each exception.

When a non-maskable data exception is recognized, a program interruption for a data exception always occurs.

Each of the IEEE exceptions is controlled by a mask bit in the floating-point-control (FPC) register. The handling of these exceptions is described in the section "IEEE Exceptions" on page 9-19.

A data exception is recognized for the following cases:

- **General-operand** data exception is recognized for the following cases:
  - An instruction which operates on decimal operands encounters invalid decimal digit or sign codes or has its operands specified improperly. The operation is suppressed, except that, for EDIT and EDIT AND MARK, it is model dependent whether the operation is suppressed or terminated. See the section "General-Operand Data Exception" on page 8-5 for details.
  - COMPRESSION CALL encounters errors in its dictionaries, in which case it is model dependent whether the operation is suppressed or terminated.
  - PERFORM RANDOM NUMBER OPERATION when the reseed counter is zero for the generate operation, in which case the operation is suppressed.
  - Execution of SORT LISTS is attempted and any of the following applies:
    1) The SORTL-SFLR or SORT-SVLR function is specified and no bits, or multiple bits, of bits 0-7 of the parameter-block-version number, contain a value of one.
    2) The SORTL-SFLR or SORTL-SVLR function is specified and the size or format of the parameter block, as specified by bits 0-7 and 12-15 of the parameter-block-version number, respectively, is not supported by the model.
    3) The SORTL-SFLR or SORTL-SVLR function is specified and the record-key length specifies a key size of zero, a key size which is not a multiple of 8, or a key size greater than 4096.
    4) The SORTL-SFLR function is specified and the record-payload length specifies a payload size which is not a multiple of 8, or a payload size, when added to the key size, is greater than 4096.
    5) The SORTL-SVLR function is specified and the record-payload length specifies a payload size which is not a multiple of 8, or a payload size, when added to the key size, is greater than 4088, in which case it is model dependent whether the operation is suppressed or terminated.
    6) The SORTL-SFLR or SORTL-SVLR function is specified and the value of the active-input-lists count code (AILCC) plus one is greater than the number of input lists described by the parameter block.
    7) The SORTL-SFLR or SORTL-SVLR function is specified and an input-list address, corresponding to an active input list, is not designated on a double-word boundary.

    When execution of SORT LISTS is attempted and a general-operand data exception is recognized, the operation is suppressed, except as described above.

  The general-operand data exception is reported with DXC 0.

  Note: In earlier versions of the architecture, the general-operand data exception was known as the decimal-operand data exception.

- **AFP-register** data exception is recognized when bit 45 of control register 0 is zero, and a floating-point-support (FPS) instruction or a hexadecimal-floating-point (HFP) instruction specifies a floating-point register other than 0, 2, 4, or 6. AFP-register data exception is also recognized when bit 45 of control register 0 is zero and a PFPO instruction is executed. The operation is suppressed and is reported with DXC 1.

- **BFP-instruction** data exception is recognized when bit 45 of control register 0 is zero and a BFP instruction is executed. The operation is suppressed and is reported with DXC 2.

- **Compare-and-trap-instruction** data exception is recognized when the operands compared by COMPARE AND TRAP, COMPARE IMMEDIATE AND TRAP, COMPARE LOGICAL AND TRAP, or

COMPARE LOGICAL IMMEDIATE AND TRAP match the conditions specified by the M_3 field of the instruction. The operation is completed and is reported with DXC FF hex.

When the load-and-trap facility is installed, compare-and-trap-instruction data exception is recognized when all zeros are loaded into the first operand of LOAD AND TRAP, LOAD HIGH AND TRAP, LOAD LOGICAL AND TRAP, and LOAD LOGICAL THIRTY ONE BITS AND TRAP. The operation is completed and is reported with DXC FF hex.

- **DFP-instruction** data exception is recognized when bit 45 of control register 0 is zero and a DFP instruction is executed. The operation is suppressed and is reported with DXC 3.
- **IEEE-exception** data exceptions are recognized when an IEEE computational instruction encounters an enabled exceptional condition. The operation is suppressed or completed, depending on the type of exception. See the section "IEEE Exceptions" on page 19-8 for details.
- **Simulated IEEE-exception** data exceptions are recognized when an IEEE-exception-simulation instruction (LOAD FPC AND SIGNAL or SET FPC AND SIGNAL) encounters an enabled signaling flag. The operation is completed. See the section "IEEE Exceptions" on page 19-8 for details.
- **Vector-instruction** data exceptions are recognized when bit 46 of control register 0 is zero and a vector instruction is executed. It is unpredictable if a data exception is recognized if bit 45 of control register 0 is zero, bit 46 of control register 0 is one, and a vector instruction is executed. The operation is suppressed and is reported with DXC FE hex.

The instruction-length code is 1, 2, or 3.

The data exception is indicated by a program-interruption code of 0007 hex (or 0087, 0207, or 0287 hex, if a concurrent PER event, a concurrent transactional-execution-aborted event, or both are indicated, respectively).

⋮

Specification Exception

A specification exception is recognized when any of the following is true:

⋮

73. Execution of SET FPC or SET FPC AND SIGNAL is attempted, and one or more bits of the first operand corresponding to unsupported bits in the FPC register are one.

74. Execution of SORT LISTS is attempted, and any of the following applies:

    - Bits 57-63 of general register 0 designate an unassigned or uninstalled function code.
    - The R_1 field designates an odd-numbered register or general register 0.
    - The R_2 field designates an odd-numbered register or general register 0.
    - The parameter block is not designated on a doubleword boundary.
    - The SORTL-SFLR function or the SORTL-SVLR function is specified and the first operand is not designated on a doubleword boundary.
    - The SORTL-SFLR function or the SORTL-SVLR function is specified, merge mode is zero, and the second operand is not designated on a doubleword boundary.

75. Execution of STORE SYSTEM INFORMATION is attempted, the function code in general register 0 is valid, and bits 36-55 of general register 0 and bits 32-47 of general register 1 are not all zeros.

⋮

Chapter 26. Specialized-Function-Assist Instructions

⋮

Instructions

⋮

Programming Notes:

1. The DEFLATE CONVERSION CALL instruction is available when the DEFLATE-conversion facility is installed.
2. The SORT LISTS instruction is available when the enhanced-sort facility is installed.

Name	Mnemonic	Characteristics							Opcode	Page
⋮										
SORT LISTS	SORTL	RRE C ES	¤[5,9] A SP	IC Dg	GM I1		ST	R_1 R_2	B938	26-2
⋮										

Explanation:

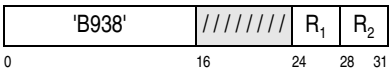
¤[5]	Model dependent whether the instruction is restricted from transactional execution.
¤[9]	Restricted in the constrained transactional-execution mode.
A	Access exceptions for logical addresses.
C	Condition code is set.
⋮	⋮
Dg	General-operand data exception.
⋮	⋮
ES	Enhanced-sort facility.
GM	Instruction execution includes the implied use of multiple general registers: • General registers 0 and 1 for DEFLATE CONVERSION CALL and SORTL LISTS.
I1	Access register 1 is implicitly designated in the access-register mode.
IC	Condition code alternative to interruptible instruction
R_1	R_1 field designates an access register in the access-register mode.
R_2	R_2 field designates an access register in the access-register mode.
R_3	R_3 field designates an access register in the access-register mode.
RRE	RRE instruction format.
RRF	RRF instruction format.
SP	Specification exception.
ST	PER storage-alteration event.

Figure 26-1. Summary of General Instructions

⋮

SORT LISTS

SORTL R_1,R_2 [RRE]



SORT LISTS performs functions related to sorting records of data.

SORT LISTS is an unprivileged instruction. It may be executed when the CPU is in the problem or supervisor state.

A function specified by the function code in general register 0 is performed.

Bits 16-23 of the instruction are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future.

Bit positions 57-63 of general register 0 contain the function code. Figure 26-2 shows the assigned function codes for SORT LISTS. All other function codes are unassigned.

Code	Function	Parameter Block Size (bytes)
0	SORTL-QAF	32
1	SORTL-SFLR	$(576+16xN_{IS})^{F0}$
2	SORTL-SVLR	$(576+16xN_{IS})^{F0}$

Explanation

F0 Format-0 parameter block (the format is specified in the PBVN field)

N_{IS} Number of input lists, as specified by the Interface Size

Figure 26-2. Function Codes for SORT LISTS

When bits 57-63 of general register 0 designate an unassigned or uninstalled function code, a specification exception is recognized.

Bit 56 of general register 0 specifies a mode of operation (merge mode) which applies to the SORTL-SFLR and SORTL-SVLR functions. For a description of merge mode (MM), refer to section "Function Code 1: SORTL-SFLR (Sort Fixed-Length-Records)" beginning on page 26-5. Bit 56 of general register 0 is ignored when the specified function is SORTL-QAF.

Bit positions 0-31 of general register 0 are ignored. Bit positions 32-55 of general register 0 are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future.

The contents of general register 1 specify the logical address of the leftmost byte of the parameter block in storage. The parameter block must be designated on a doubleword boundary; otherwise a specification exception is recognized.

For all functions, the contents of general registers 0 and 1 are not modified.

For all functions, the R_1 field designates an even-odd pair of general registers. For all functions, the R_1 field must not designate general register 0 and must designate an even-numbered register; otherwise, a specification exception is recognized. When the specified function is SORTL-SFLR or SORTL-SVLR, the contents of general register R_1 specify the logical address of the leftmost byte of the first operand and the contents of general register $R_1 + 1$ specify the length of the first operand in bytes. When the specified function is SORTL-SFLR or SORTL-SVLR, the first operand must be designated on a doubleword boundary; otherwise a specification exception is recognized. Data, in the form of records, is selected from a set of input lists and is stored at the first-operand location. When the SORTL-QAF function is specified, the contents of general registers R_1 and $R_1 + 1$ are ignored.

For all functions, the R_2 field designates an even-odd pair of general registers. For all functions, the R_2 field must not designate general register 0 and must designate an even-numbered register; otherwise, a specification exception is recognized. When the specified function is SORTL-SFLR or SORTL-SVLR, and merge mode (MM) is zero, the contents of general register R_2 specify the logical address of the leftmost byte of the second operand and the contents of general register $R_2 + 1$ specify the length of the second operand in bytes. When the specified function is SORTL-SFLR or SORTL-SVLR, and merge mode (MM) is zero, the second operand must be designated on a doubleword boundary; otherwise a specification exception is recognized. The starting address and length of each output list, referred to as output-list delineations (OLD), are stored at the second-operand location when MM is zero. When the

SORTL-QAF function is specified, or MM is one, the contents of general registers R_2 and $R_2 + 1$ are ignored.

As part of the operation when the specified function is SORTL-SFLR or SORTL-SVLR, the following occurs:

- The address in general register R_1 is incremented by the number of bytes stored at the first-operand location, and the length in general register $R_1 + 1$ is decremented by the same number.
- When MM is zero, the address in general register R_2 is incremented by the number of bytes stored at the second-operand location, and the length in general register $R_2 + 1$ is decremented by the same number.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

In the 24-bit addressing mode, the following apply:

- The contents of bit positions 40-63 of general registers 1, R_1, and R_2 constitute the addresses of the parameter block, first operand, and second operand, respectively, and the contents of bit positions 0-39 are ignored.
- Bits 40-63 of the updated first-operand and second-operand addresses replace the corresponding bits in general registers R_1 and R_2, respectively. Carries out of bit position 40 of the updated addresses are ignored, and the contents of bit positions 32-39 of general registers R_1 and R_2 are set to zeros. The contents of bit positions 0-31 of general registers R_1 and R_2 remain unchanged.
- The contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$ form 32-bit unsigned binary integers which specify the number of bytes in the first and second operands, respectively. The contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_2 + 1$ are ignored.
- Bits 32-63 of the updated first-operand and second-operand lengths replace the corresponding bits in general registers $R_1 + 1$ and $R_2 + 1$, respectively. The contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

In the 31-bit addressing mode, the following apply:

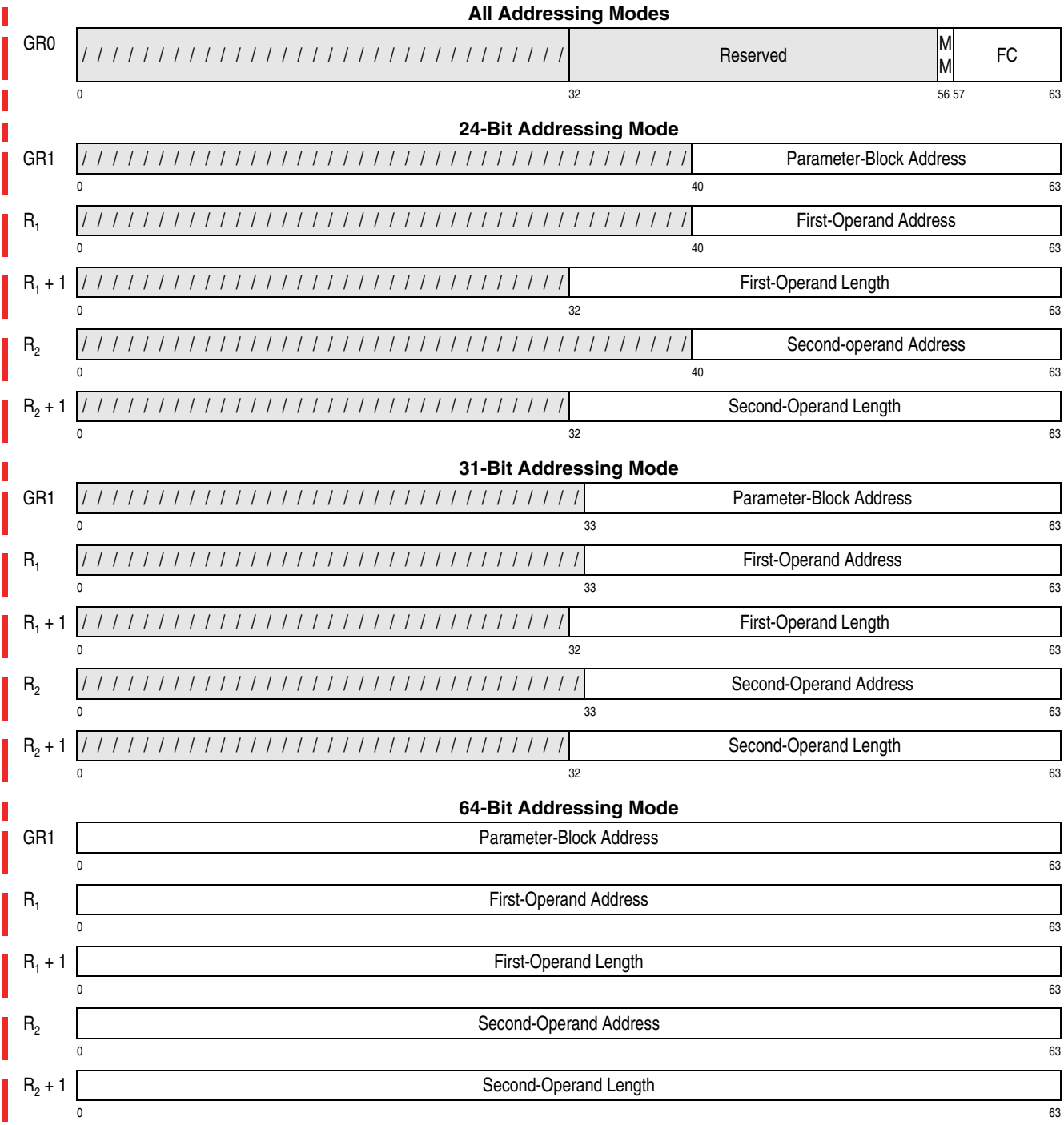
- The contents of bit positions 33-63 of general registers 1, R_1, and R_2 constitute the addresses of the parameter block, first operand, and second operand, respectively, and the contents of bit positions 0-32 are ignored.
- Bits 33-63 of the updated first-operand and second-operand addresses replace the corresponding bits in general registers R_1 and R_2, respectively. Carries out of bit position 33 of the updated addresses are ignored, and the content of bit position 32 of general registers R_1 and R_2 is set to zero. The contents of bit positions 0-31 of general registers R_1 and R_2 remain unchanged.
- The contents of bit positions 32-63 of general registers $R_1 + 1$ and $R_2 + 1$ form 32-bit unsigned binary integers which specify the number of bytes in the first and second operands, respectively. The contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_2 + 1$ are ignored.
- Bits 32-63 of the updated first-operand and second-operand lengths replace the corresponding bits in general registers $R_1 + 1$ and $R_2 + 1$, respectively. The contents of bit positions 0-31 of general registers $R_1 + 1$ and $R_2 + 1$ remain unchanged.

In the 64-bit addressing mode, the following apply:

- The contents of bit positions 0-63 of general registers 1, R_1, and R_2 constitute the addresses of the parameter block, first operand, and second operand, respectively.
- Bits 0-63 of the updated first-operand and second-operand addresses replace the corresponding bits in general registers R_1 and R_2, respectively. Carries out of bit position 0 of the updated addresses are ignored.
- The contents of bit positions 0-63 of general registers $R_1 + 1$ and $R_2 + 1$ form 64-bit unsigned binary integers which specify the number of bytes in the first and second operands, respectively.
- Bits 0-63 of the updated first-operand and second-operand lengths replace the corresponding bits in general registers $R_1 + 1$ and $R_2 + 1$, respectively.

In the access-register mode, access registers 1, R_1, and R_2 specify the address spaces containing the parameter block, first operand, and second operand, respectively.

Figure 26-3 on page 26-4 shows the contents of the general registers just described.



Explanation:

FC Function code.
MM Merge Mode.

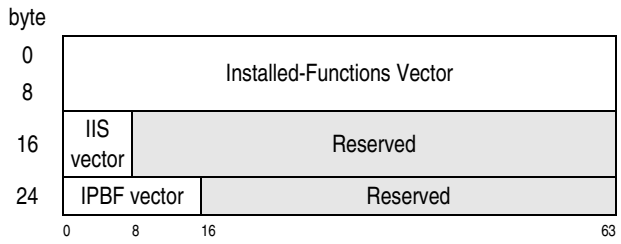
Figure 26-3. General Register Assignments for SORTL

Function Code 0: SORTL-QAF (Query Available Functions)

The SORTL-QAF (query) function provides the means of indicating the availability of all installed

functions, installed parameter block formats, and interface sizes available. An interface size is the number of input lists available to the program. The size of the parameter block for the SORT-SFLR and SORT-SVLR functions is proportional to the interface size specified by the program.

The parameter block for the SORTL-QAF function has the following format:



Explanation

IIS Installed-interface-sizes

IPBF Installed-parameter-block-formats

Figure 26-4. Parameter Block for SORTL-QAF

An installed-functions vector, an installed-interface-sizes vector, and an installed-parameter-block-formats vector are stored to bytes 0-15, byte 16, and bytes 24-25, respectively, of the parameter block, as illustrated in Figure 26-4.

Bits 0-127 of the installed-functions vector correspond to function codes 0-127, respectively, of the SORT LISTS instruction. When a bit is one, the corresponding function is installed; otherwise, the function is not installed.

Bits 0-7 of the installed-interface-sizes vector indicate the interface sizes available to the program. An interface size is the number of input lists to be specified by the program for the SORT-SFLR and SORTL-SVLR functions. Bits 0-7 of the installed-interface-sizes vector correspond to the following interface sizes:

bit	interface size
0	reserved
1	reserved
2	32 input lists
3	64 input lists
4	128 input lists
5	reserved
6	reserved
7	reserved

When a bit is one, the corresponding interface size is available to the program. One or more bits may be stored as ones. For example, a value of 00101000 binary indicates interfaces sizes of 32 and 128 input lists are available. Bits 0-1 and 5-7 are reserved and stored as zeros. The interface size of 32 input lists is always available when the enhanced-sort facility is installed. Therefore, bit 2 is always stored as one.

Bits 0-15 of the installed-parameter-block-formats vector correspond to parameter-block formats 0-15, respectively for the SORTL-SFLR and SORTL-SVLR functions. When a bit is one, the corresponding parameter-block format is installed; otherwise, the parameter-block format is not installed.

Zeros are stored to reserved bytes 17-23 and 26-31 of the parameter block.

The contents of general registers R_1, R_2, $R_1 + 1$, and $R_2 + 1$ are ignored by the SORT-QAF function.

A PER storage-alteration event is recognized, when applicable, for the parameter block. A PER zero-address-detection event is recognized, when applicable, for the parameter block.

Condition code 0 is set when execution of the SORTL-QAF function completes; condition codes 1, 2, and 3 are not applicable to the query function.

Function Code 1: SORTL-SFLR (Sort Fixed-Length-Records)

A set of input lists is sorted and stored as a set of output lists at the first-operand location. Each list is a set of records. Each record consists of a fixed-length key and a fixed-length payload, as shown in Figure 26-5 on page 26-5.

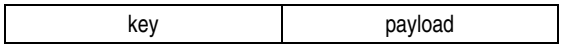


Figure 26-5. Fixed-Length-Record format

Records from the input lists are sorted based on the values of the keys. The records may be sorted in

ascending, or descending order, as specified in the sort order (SO) field of the parameter block.

An input list is referred to as being presorted in ascending order when the key of each record of the input list is greater than, or equal to, the key of the adjacent record on the left in the same input list. An input list is referred to as being presorted in descending order when the key of each record of the input list is less than, or equal to, the key of the adjacent record on the left in the same input list.

The records of an output list may be sourced from multiple input lists, and are stored in sorted order. The number of output lists stored at the first-operand location depends on the input data. When every active input list is presorted in the same order as specified in the SO field, only one output list is produced.

Bit 56 of general register 0 specifies a mode of operation, referred to as merge mode (MM), which distinguishes whether or not more information related to the results stored at the first-operand location is provided. MM applies to the SORTL-SFLR function.

When merge mode is zero, no active input list is considered presorted. When MM is zero, for each output list stored at the first-operand location, a corresponding output-list delineation (OLD) is stored at the second-operand location. Each OLD consists of an 8-byte OLD-address, which designates the location of the first record in the corresponding output list, followed by an 8-byte OLD-length, which specifies the length, in bytes, of the corresponding output list.

When merge mode is one, every active input list is considered presorted in the same order as specified by the SO field of the parameter block.

When merge mode is one and every active input list is presorted in the same order as specified by the SO field of the parameter block, the result stored at the first-operand location is a single output list of records in sorted order. When MM is one and any active input list is not presorted in the same order as specified by the SO field of the parameter block, results are unpredictable.

When merge mode is one, the contents of general registers R_2 and $R_2 + 1$ are ignored and no information is stored at the second-operand location. When MM is one, procedures required to distinguish separations between output lists may not be performed, thereby potentially improving the performance of the operation. When MM is one, data is not stored to the continuation-record-recall buffer.

The locations of the operands and addresses used by the SORTL-SFLR function are as shown in Figure 26-3 on page 26-4.

SORTL-SFLR: Parameter Block

The parameter block with format 0 hex for the SORTL-SFLR function is shown in Figure 26-6 on page 26-6. The format is specified in the PBVN field of the parameter block.

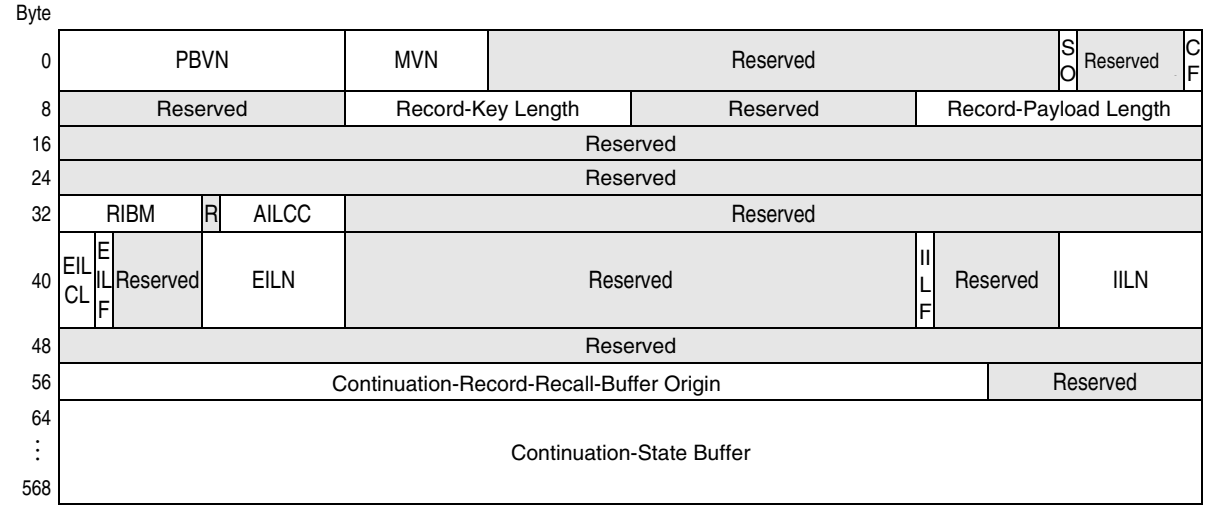
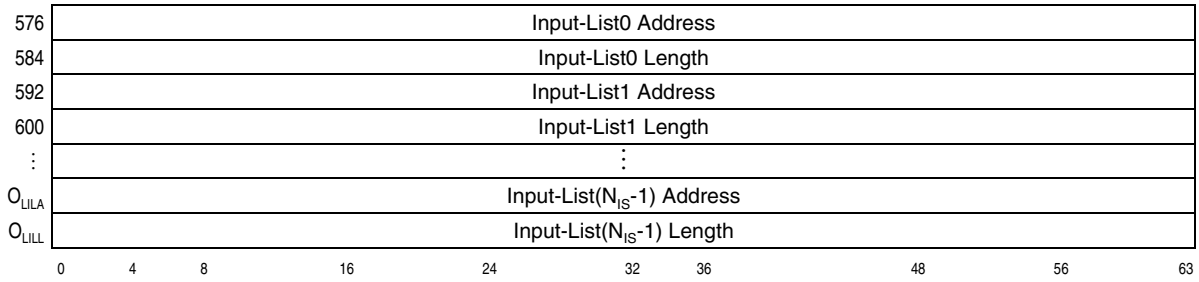


Figure 26-6. Parameter Block (format 0 hex) for SORTL-SFLR and SORTL-SVLR (Part 1 of 2)



Explanation:

AILCC	Active-Input-Lists Count Code
CF	Continuation Flag
EILCL	Empty-Input-Lists Control
EILF	Empty-Input-List Flag
EILN	Empty-Input-List Number
IILF	Incomplete-Input-List Flag
IILN	Incomplete-Input-List Number
MVN	Model-Version Number
N_{IS}	Number of input lists, as specified by the Interface Size
O_{LILA}	Offset of the Last Input List Address: $O_{LILA} = 576 + 16x(N_{IS} - 1)$
O_{LILL}	Offset of the Last Input List Length: $O_{LILL} = 576 + 16x(N_{IS} - 1) + 8$
PBVN	Parameter-Block-Version Number
R	Reserved
RIBM	Reserved for IBM use
SO	Sort Order

Figure 26-6. Parameter Block (format 0 hex) for SORTL-SFLR and SORTL-SVLR (Part 2 of 2)

The fields of the format-0 parameter block for the SORTL-SFLR function are defined as follows:

Reserved: As an input to the operation, reserved fields should contain zeros; otherwise, the program may not operate compatibly in the future. When the operation ends, reserved fields may be stored as zeros or may remain unchanged.

Parameter-Block-Version Number (PBVN): Bytes 0-1 of the parameter block specify the version and size of the parameter block. Bits 0-7 of the PBVN have the same format and definition as bits 0-7 of the installed-interface-sizes vector (byte 16) of the parameter block for the SORTL-QAF (query) function. Bits 0-7 specify the number of input lists described in the parameter block, N_{IS}. The size of the format-0 parameter block, in bytes, is determined by evaluating the formula ($576 + 16xN_{IS}$). One and only one bit of bits 0-7 must have a value of one; otherwise a general-operand data exception is recognized. Bits 8-11 of the PBVN are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future. Bits 12-15 of the PBVN contain an unsigned binary integer specifying the format of the parameter block. The SORTL-QAF function provides the means of indicating the parameter block formats available. When the size or format of the parameter block specified is not supported by the model, a general-operand data exception is recognized. The PBVN is specified by the program and is not modified during the execution of the instruction.

Model-Version Number (MVN): Byte 2 of the parameter block is an unsigned binary integer identifying the model which executed the instruction. The MVN is updated during the execution of the instruction. The value stored in the MVN is model-dependent.

When the continuation flag (CF) is one, the MVN is an input to the operation. When CF is one and the MVN identifies the same model as the model currently executing the instruction, data from the continuation-state buffer (CSB) may be used to resume the operation. When CF is one and the MVN identifies a different model than the model currently executing the instruction, part, or all of the CSB field may be ignored.

The program must not modify the MVN in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Sort Order (SO): Bit 56 of the parameter block, when zero, specifies an ascending sort order, and when one, specifies a descending sort order. When ascending sort order is specified, each record of an output list contains a key that is greater than, or equal to, the key of the adjacent record on the left, in the same output list. When descending sort order is specified, each record of an output list contains a key that is less than, or equal to, the key of the adjacent record on the left, in the same output list. The SO is not updated during the execution of the instruction.

Continuation Flag (CF): Bit 63 of the parameter block, when one, indicates the operation is partially complete and the contents of the continuation-state buffer (CSB), and when merge mode (MM) is zero, the contents of the continuation-record-recall buffer, may be used to resume the operation. It is required for the program to initialize the continuation flag (CF) to zero and not modify CF in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Record-Key Length: Bytes 10-11 of the parameter block contain an unsigned binary integer specifying the size, in bytes, of all keys, in all records processed during the operation. A general-operand data exception is recognized for any of the following conditions:

- A key size of zero bytes is specified.
- A key size which is not a multiple of 8 is specified.
- A key size larger than 4096 bytes is specified.

The record-key length is not updated during the execution of the instruction.

Record-Payload Length: When the SORTL-SFLR function is specified, bytes 14-15 of the parameter block contain an unsigned binary integer specifying the size, in bytes, of all payloads, in all records processed during the operation. A general-operand data exception is recognized for any of the following conditions:

- A payload size which is not a multiple of 8 is specified.
- The sum of the key and payload sizes specified is larger than 4096 bytes.

A payload size of zero is valid.

When the SORTL-SVLR function is specified, the record-payload length field of the parameter block is ignored. The record-payload length is not updated during the execution of the instruction.

Reserved for IBM use (RIBM): Byte 32 of the parameter block is reserved for IBM use and must contain zeros; otherwise results are unpredictable. The RIBM field is not modified during the execution of the instruction.

Active-Input-Lists Count Code (AILCC): Bits 1-7 of byte 33 of the parameter block are a 7-bit unsigned integer that specifies the number of the input list which denotes the boundary between active and inactive input lists. Input lists with list numbers less than or equal to the value of the AILCC field are in the active state. Input lists with list numbers greater than the value of the AILCC field are in the inactive state. The number of input lists in the active state is one more than the value in the AILCC field.

Input lists in the active state participate in the operation. Input lists in the inactive state do not participate in the operation.

When the value of the AILCC field plus one is greater than the number of input lists described in the parameter block, as specified by bits 0-7 of the PBVN field, a general-operand data exception is recognized.

The value specified in the AILCC field does not affect the size of the parameter block. Access exceptions apply to the entire parameter block, regardless of any input list being in the inactive state.

The AILCC is not updated during the execution of the instruction.

Empty-Input-Lists Control (EILCL): When bit 0 of byte 40 of the parameter block is one, the operation ends when the length of input list0 becomes zero during the operation. When bit 0 of byte 40 of the parameter block is zero, the operation continues to proceed when the length of input list0 becomes zero during the operation. When bit 1 of byte 40 of the parameter block is one, the operation ends when the length of an active input list, other than input list0, becomes zero during the operation. When bit 1 of byte 40 of the parameter block is zero, the operation continues to proceed when the length of an active input list, other than input list0, becomes zero during

the operation. See programming note 1 for a description of the intended values and uses of the EILCL field.

When the length of an active input list is initially zero before execution of the instruction, the corresponding bit of the EILCL does not apply.

The EILCL is not updated during the execution of the instruction.

The program must not modify the EILCL in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Empty-Input-List Flag (EILF): When the EILCL is 11 binary, and the operation ends due to the updated length of an active input list is equal to zero, and condition code 2 is set, the value of one is stored to bit 2, of byte 40, of the parameter block; otherwise the value of zero is stored to bit 2, of byte 40, of the parameter block. When the EILF contains a value of one, the input list number of the input list which became empty during the operation is placed in the EILN field of the parameter block.

The EILF may be referenced at the beginning of the execution of the instruction when the operation is being resumed. The program must not modify the EILF in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Empty-Input-List Number (EILN): When conditions cause a value of one to be stored in the EILF field, the input list number of the input list which became empty during the operation is stored in byte 41 of the parameter block; otherwise the value of zero is stored in byte 41 of the parameter block.

The EILN is ignored at the beginning of the operation.

Incomplete-Input-List Flag (IILF): When the operation ends as a result of attempting to process an incomplete input list, the value of one is stored to bit 0, of byte 46, of the parameter block; otherwise the value of zero is stored to bit 0, of byte 46, of the parameter block. An active input list is considered to be incomplete when the corresponding input-list length is greater than zero and less than the number of bytes of the record designated by the input-list address. This condition may exist at the beginning of the operation, or may be encountered during the operation. When the IILF contains a value of one, the input list number, of the incomplete input list encountered, is placed in the IILN field of the parameter block.

When the operation ends with setting condition code 2 and the resulting value in the IILF field is zero, the operation ended due to an empty input list. When the operation ends with setting condition code 2 and the resulting value in the IILF field is one, the operation ended due to an incomplete input list.

The IILF may be referenced at the beginning of the execution of the instruction when the operation is being resumed. The program must not modify the IILF in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Incomplete-Input-List Number (IILN): When conditions cause a value of one to be stored in the IILF field, the input list number, of the incomplete input list encountered, is stored in byte 47 of the parameter block; otherwise the value of zero is stored in byte 47 of the parameter block. When multiple input lists are incomplete, it is model dependent which incomplete input list number is stored to the IILN field.

The IILN is ignored at the beginning of the operation.

Continuation-Record-Recall-Buffer Origin: A 4 K-byte buffer in storage, called the continuation-record-recall buffer, is provided by the program for the CPU to store and reference data between two executions of the same SORT LISTS instruction, in case an operation ends and may be resumed later. Bit 0 of byte 56 through bit 3 of byte 62 of the parameter block specify bits 0-51 of the continuation-record-recall-buffer origin, which is used in the formation of the continuation-record-recall address. The continuation-record-recall address is aligned on a 4 K-byte boundary and is the logical address of the leftmost byte of the continuation-record-recall buffer.

In the 24-bit addressing mode, bits 40-51 of the continuation-record-recall-buffer origin with 12 zeros appended to the right form the continuation-record-recall address. In the 31-bit addressing mode, bits 33-51 of the continuation-record-recall-buffer origin with 12 zeros appended to the right form the continuation-record-recall address. In the 64-bit addressing mode, bits 0-51 of the continuation-record-recall-buf-

fer origin with 12 zeros appended to the right form the continuation-record-recall address.

In the access-register mode, access register 1 specifies the address space containing the continuation-record-recall buffer in storage.

When merge mode (MM) is zero and the operation ends after storing one or more records, the key of the last record stored to the first operand may also be stored to the continuation-record-recall buffer.

When MM is one, the continuation-record-recall-buffer origin is ignored.

The continuation-record-recall-buffer origin is not modified during the execution of the instruction.

The program must not modify the continuation-record-recall-buffer origin in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Continuation-State Buffer (CSB): When conditions cause a value of one to be stored in the CF field, internal-state data is stored to bytes 64-575 of the parameter block; otherwise bytes 64-575 of the parameter block are undefined and may be modified. The internal-state data stored is model-dependent and may be used subsequently to resume the operation when the instruction is reexecuted. The program must not modify the continuation-state buffer in the event the instruction is to be reexecuted for the purpose of resuming the operation; otherwise results are unpredictable.

Input-ListN Address: The parameter block defines multiple input lists. The number of input lists defined in the parameter block, N_{IS}, is specified by bits 0-7 of the PBVN. The input lists are numbered from zero to (N_{IS}-1). For each input list, the parameter block specifies an 8 byte input-list address. For input list number N, the contents of bytes 576 + 16xN through 583 + 16xN, of the format-0 parameter block, specify the logical address of the leftmost byte of input list number N in storage.

Each input-list address corresponding to an input list in the active state, as specified by the AILCC field, is an input to the operation and is updated by the operation. Each input-list address corresponding to an input list in the inactive state, as specified by the AILCC field, is ignored by the operation.

When an input-list address is an input to the operation, the following applies:

- In 24-bit addressing mode, bits 40-63, of the input-list address, designate the location of the leftmost byte of the input list in storage, and the contents of bits 0-39, of the input-list address are treated as zeros.
- In 31-bit addressing mode, bits 33-63, of the input-list address, designate the location of the leftmost byte of the input list in storage, and the contents of bits 0-32, of the input-list address are treated as zeros.
- In 64-bit addressing mode, bits 0-63, of the input-list address, designate the location of the leftmost byte of the input list in storage.

In the access-register mode, access register 1 specifies the address space containing all active input lists in storage.

For all input lists in the active state, the corresponding input-list address must be designated on a doubleword boundary; otherwise, a general-operand data exception is recognized.

When an input-list address is updated by the operation, the following applies:

- When one or more records of the input list have been processed as part of the operation, the corresponding input-list address is incremented by the number of bytes which the processed records occupy in storage. The formation and updating of the input-list address are dependent on the addressing mode.
- In 24-bit addressing mode, bits 40-63 of the updated input-list address replace the corresponding bits in the input-list address field of the parameter block, a carry out of bit position 40 of the updated input-list address is ignored, and the contents of bit positions 0-39 of the input-list address field of the parameter block are set to zeros.
- In 31-bit addressing mode, bits 33-63 of the updated input-list address replace the corresponding bits in the input-list address field of the parameter block, a carry out of bit position 33 of the updated input-list address is ignored, and the contents of bit positions 0-32 of the input-list address field of the parameter block are set to zeros.
- In 64-bit addressing mode, bits 0-63 of the updated input-list address replace the corresponding bits in the input-list address field of the

parameter block, and a carry out of bit position 0 of the updated input-list address is ignored.

In 24- and 31-bit addressing modes, when the execution of the instruction ends and the instruction is not suppressed, nullified, or terminated, each 64-bit input-list address corresponding to an active input list is updated, even when the address is not incremented.

Input-ListN Length: For each input list, the parameter block specifies an 8 byte input-list length. For input list number N, bytes 584 + 16xN through 591 + 16xN, of the parameter block, contain an unsigned integer which specifies the number of bytes in input list number N.

Each input-list length corresponding to an input list in the active state, as specified by the AILCC field, is an input to the operation and is updated by the operation. Each input-list length corresponding to an input list in the inactive state, as specified by the AILCC field, is ignored by the operation.

In all addressing modes, the contents of bit positions 0-63 of an input-list length field specify the length of the corresponding input list.

When one or more records of an input list have been processed as part of the operation, the corresponding input-list length is decremented by the number of bytes which the processed records occupy in storage. In all addressing modes, bits 0-63 of an updated input-list length replace bits 0-63 in the corresponding input-list length field of the parameter block.

SORTL-SFLR: Ending Conditions

Normal completion occurs when all records from all active input lists have been sorted and stored to the first operand.

When the operation ends due to normal completion, the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated when MM is zero.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to zero.
- The empty-input-list flag (EILF) is set to zero.
- The empty-input-list number is set to zero.
- The incomplete-input-list flag (IILF) is set to zero.
- The incomplete-input-list number is set to zero.
- Condition code 0 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

When normal completion occurs, the CSB field of the parameter block is undefined after the operation ends.

When a CPU-determined number of bytes have been processed, the operation ends and the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated when MM is zero.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to one.
- A key value is stored to the continuation-record-recall buffer when MM is zero and one or more records have been placed at the first-operand location during the execution of the instruction.
- The continuation-state buffer is updated.
- The empty-input-list flag (EILF) is set to zero.
- The empty-input-list number is set to zero.
- The incomplete-input-list flag (IILF) is set to zero.
- The incomplete-input-list number is set to zero.
- Condition code 3 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

The CPU-determined number of bytes depends on the model, and may be a different number each time the instruction is executed. The CPU-determined number of bytes is typically nonzero. Although this number may be zero and appear as a no-progress case, the CPU protects against endless recurrence of this no-progress case.

Subsequent to the instruction ending with condition code 3 set, it is expected the program does not modify any input or output specification for the instruction

and branches back to reexecute the instruction to resume the operation.

When bit 0 of the empty-input-lists control (EILCL) is one and the length of input list0 becomes zero during the operation and normal completion does not apply, the operation ends and the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated when MM is zero.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to one.
- A key value may be stored to the continuation-record-recall buffer when EILCL is 10 binary and MM is zero. A key value is stored to the continuation-record-recall buffer when EILCL is 11 binary and MM is zero. In either case, one or more records have been placed at the first-operand location during the execution of the instruction.
- The continuation-state buffer is updated.
- The empty-input-list flag is set (refer to Figure 26-7).
- The empty-input-list number is set (refer to Figure 26-7).
- The incomplete-input-list flag is set to zero.
- The incomplete-input-list number is set to zero.
- Condition code 2 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

When bit 1 of the empty-input-lists control (EILCL) is one and the length of an active input list, other than input list0, becomes zero during the operation and normal completion does not apply, the operation ends and the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated when MM is zero.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to one.
- A key value may be stored to the continuation-record-recall buffer when EILCL is 01 binary and MM is zero. A key value is stored to the continuation-record-recall buffer when EILCL is 11 binary and MM is zero. In either case, one or more records have been placed at the first-operand location during the execution of the instruction.
- The continuation-state buffer is updated.
- The empty-input-list flag is set (refer to Figure 26-7).
- The empty-input-list number is set (refer to Figure 26-7).
- The incomplete-input-list flag is set to zero.
- The incomplete-input-list number is set to zero.
- Condition code 2 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

When an incomplete input list in the active state is encountered, the operation ends and the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated when MM is zero.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to one.
- A key value is stored to the continuation-record-recall buffer when MM is zero and one or more records have been placed at the first-operand location during the execution of the instruction.
- The continuation-state buffer is updated.
- The empty-input-list flag (EILF) is set to zero.
- The empty-input-list number is set to zero.
- The incomplete-input-list flag (IILF) is set to one.
- The input-list number of the incomplete input list encountered is placed in the incomplete-input-list number (IILN) field of the parameter block.
- Condition code 2 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

When the length of the first operand is insufficient to store another record, the operation ends and the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated when MM is zero.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to one.
- A key value may be stored to the continuation-record-recall buffer when MM is zero and one or more records have been placed at the first-operand location during the execution of the instruction.
- The continuation-state buffer is updated.
- The empty-input-list flag is set to zero.
- The empty-input-list number is set to zero.
- The incomplete-input-list flag is set to zero.
- The incomplete-input-list number is set to zero.
- Condition code 1 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

When merge mode (MM) is zero and the length of the second operand is less than 16, the operation ends and the following occurs:

- The address and length in general registers R_1 and $R_1 + 1$, respectively, are updated.
- The address and length in general registers R_2 and $R_2 + 1$, respectively, are updated.
- The input-listN address and input-listN length fields are updated for all input lists in the active state.
- The model-version number is set.
- The continuation flag is set to one.
- A key value may be stored to the continuation-record-recall buffer when one or more records have been placed at the first-operand location during the execution of the instruction.
- The continuation-state buffer is updated.
- The empty-input-list flag is set to zero.
- The empty-input-list number is set to zero.
- The incomplete-input-list flag is set to zero.
- The incomplete-input-list number is set to zero.
- Condition code 1 is set.

The formation and updating of the addresses and lengths are dependent on the addressing mode.

EILCL (binary)	condition causing operation to end	results				
		CC	IILF	IILN	EILF	EILN
—	normal completion	0	0	0	0	0
—	first or second operand length is insufficient	1	0	0	0	0
—	IL0 determined to be incomplete	2	1	0 (IL0)	0	0
—	ILN determined to be incomplete	2	1	N (ILN)	0	0
10	IL0 became empty (ILN may also be empty)	2	0	0	0	0
01	ILN became empty (IL0 may also be empty)	2	0	0	0	0
11	IL0 became empty (only one input list became empty)	2	0	0	1	0 (IL0)
11	ILN became empty (only one input list became empty)	2	0	0	1	N (ILN)
—	CPU-determined number of bytes processed	3	0	0	0	0

Explanation:

—	any value
IL0	input list with list number 0
ILN	input list with list number N, where N > 0

Figure 26-7. SORTL Parameter Block Fields IILF, IILN, EILF, and EILN when operation ends

The operation ending condition is called partial completion when the execution of the instruction ends in completion (does not end in suppression, nullification, or termination) and normal completion does not occur.

A PER storage-alteration event is recognized, when applicable, for the following:

- Stores to the parameter block, as described below.
- Stores to the first-operand location.
- Stores to the second-operand location.
- Stores to the continuation-record-recall buffer.

When the entire parameter block lies within the PER storage-area designation, a PER storage-alteration

event is recognized, when applicable, for the parameter block. When only a portion of the parameter block lies within the PER storage-area designation, it is model-dependent which of the following occurs:

- A PER storage-alteration event is recognized, when applicable, for the entire parameter block.
- A PER storage-alteration event is recognized, when applicable, for the portion of the parameter block that is stored.

When a PER storage-alteration event is recognized, fewer than 4K additional bytes are stored to the operand location which lies within the designated PER storage-area, before the event is reported.

A PER zero-address-detection event is recognized, when applicable, for the parameter block, first-operand location, and second-operand location. Zero-address detection does not apply to the input list addresses and the continuation-record-recall-buffer origin, which are specified in the parameter block.

As part of the operation when merge mode is zero or one, the input-list addresses and lengths for all input lists in the active state are updated. For each input-list in the active state, the input-list address is incremented by the number of bytes of all records from the input list which were selected and placed at the first-operand location during the operation, and the input-list length is decremented by the same number. The formation and updating of the input-list addresses are dependent on the addressing mode.

As the operation proceeds, an incomplete input list may be encountered. An incomplete input list is recognized during a unit of operation which attempts to reference a record from an input list which is incomplete. Multiple units of operation may be completed prior to recognizing an incomplete input list. This applies when merge mode is zero or one.

As the operation proceeds, an access exception for an access to an active input list, the first operand, or the second operand, when applicable, may be encountered. An access exception is recognized during a unit of operation which attempts to access a storage location and an access exception exists for that location. Multiple units of operation may be completed prior to recognizing an access exception. This applies when merge mode is zero or one. Refer to Figure 26-15 on page 26-24 to determine which access-exception condition is recognized when multiple conditions exist concurrently.

When the operation ends with partial completion, internal-state data, which may include a history of prior comparisons between records, is stored to the continuation-state buffer (CSB) field of the parameter block. Subsequently, when the instruction is reexecuted, for the purpose of resuming the operation, the contents of the CSB may be loaded into the CPU and the history may be referenced when the operation resumes. This applies when merge mode is zero or one.

Refer to section “Other Conditions”, beginning on page 26-22, for descriptions of other conditions that apply to the SORTL-SFLR function.

SORTL-SFLR: Processing

The SORTL-SFLR function consists of selecting records from a set of input lists, in the sort order specified, and placing the selected records at the first-operand location. As the operation proceeds, current values for the first-operand address and addresses for all active input lists are maintained. The function proceeds in units of operation. During each unit of operation, for each active input list, the key designated by the corresponding current input-list address is examined and one record is placed at the first-operand location.

The SORTL-SFLR function includes multiple comparisons between keys of records from different input lists. When comparing keys, the following applies:

- Keys are treated as unsigned-binary integers, also referred to as unstructured data.
- When comparing keys of equal value, the key from the input list with the highest input-list number is selected to be in sort order before other keys with the same value. In this case, the corresponding record from the input list with the highest input-list number is stored to the first operand before other records with the same key value. This applies for ascending and descending sort orders.

When merge mode (MM) is zero, the active input lists designate lists, each of which is treated as containing records, from left to right, in random order. When MM is zero, the records stored to the first-operand location constitute one or more output lists, and the starting address and length of each output list is stored to the second-operand location. When MM is zero, each unit of operation consists of the following steps, in the order specified:

1. Determine if the next record to store to the first-operand location may be included in the most recent output list (the output list which includes the record most recently stored to the first-operand location), as follows:

   - When the continuation flag (CF) is zero and the first unit of operation is being processed, no records have been stored to the first-operand location, and the next record to store will be the first record of an output list.
   - When CF is one, the prior execution of the instruction ended with condition code 1, and the first unit of operation is being processed for the current execution of the instruction, the next record to store will be the first record of an output list.
   - When CF is one, IILF is zero, EILF is zero, the prior execution of the instruction ended with condition code 2, and the first unit of operation is being processed for the current execution of the instruction, the next record to store will be the first record of an output list.
   - When CF is one, IILF or EILF is one, the prior execution of the instruction ended with condition code 2, and the first unit of operation is being processed for the current execution of the instruction, the next record to store may be included in the most recent output list.
   - When CF is one, the prior execution of the instruction ended with condition code 3, and the first unit of operation is being processed for the current execution of the instruction, the next record to store may be included in the most recent output list.
   - When the unit of operation being processed is not the first unit of operation for the current execution of the instruction, the next record to store may be included in the most recent output list.

   Note: The CPU stores the presence or absence of certain conditions to the continuation-state buffer when the execution of the instruction ends with partially completing an operation. The CPU references this information when the operation is subsequently resumed.

2. When the next record to store may be included in the most recent output list, determine the set of records which qualify to be included in the most recent output list. For each input list which is active, not empty and not incomplete, compare the key of the record designated by the current input-list address (current input key) to the key of the record most recently stored to the first-operand location (previously stored key). For this purpose, the reference to the previously stored key is not a reference to the first-operand location. Instead, it is a reference to the input list from which the key was previously selected, or it is a reference to the continuation-record-recall buffer. It is a reference to the continuation-record-recall buffer when the operation is being resumed and the current execution of the instruction has not yet placed any records at the first-operand location.

   When the sort order is ascending and the value of the current input key is greater than or equal to the value of the previously stored key, consider the current input key as belonging to a set of keys qualifying for inclusion in the most recent output list. When the sort order is descending and the value of the current input key is less than or equal to the value of the previously stored key, consider the current input key as belonging to a set of keys qualifying for inclusion in the most recent output list. When the number of keys in the set of keys qualifying for inclusion in the most recent output list is zero, the next record to store will be the first record of an output list. When the number of keys in the set of keys qualifying for inclusion in the most recent output list is nonzero, the next record to store will be included in the most recent output list.

3. When the next record to store will be included in the most recent output list, compare all keys in the set of keys qualifying for inclusion in the most recent output list. When the sort order is ascending, select the smallest key value and corresponding record. When the sort order is descending, select the largest key value and corresponding record. Refer to page 26-14 for the result of comparing keys of equal value.

4. When the next record to store will be the first record of an output list, compare the keys of the records designated by all current input-list addresses corresponding to input lists which are active, not empty, and not incomplete. When the sort order is ascending, select the smallest key value and corresponding record. When the sort order is descending, select the largest key value and corresponding record.

5. The selected record is placed at the current first-operand location.

6. The current first-operand address is incremented by the number of bytes equal to the length of the selected record.

7. The current input-list address, corresponding to the input list containing the selected record, is incremented by the number of bytes equal to the length of the selected record.

As part of the operation when merge mode is zero, for each output list stored at the first-operand location, a corresponding output-list delineation (OLD) is stored at the second-operand location. Each OLD consists of an 8-byte OLD-address, which designates the location of the first record in the corresponding output list, followed by an 8-byte OLD-length, which specifies the length, in bytes, of the corresponding output list. When the operation ends with condition code 3, or condition code 2 and EILF equal to one, or condition code 2 and IILF equal to one, the most recent output list being processed at the end of the operation may be partially processed and not completely processed. That is, the number of records in the partially processed output list is an intermediate value and may be increased when the operation resumes. In this case, an output-list delineation (OLD), corresponding to the partially processed output list, is not placed at the second-operand location, until after the operation is resumed and the output list is completely processed.

When merge mode is zero and the operation ends after storing one or more records and normal completion does not occur, the key of the last record stored to the first-operand location is also stored to the continuation-record-recall buffer in the cases described in the definition of the continuation-record-recall-buffer origin on page 26-9.

When merge mode is zero and the operation ends due to normal completion, one or more output lists have been placed at the first-operand location and output-list delineations have been placed at the second-operand location. The program may use output-list delineations as input-list address and length values in a parameter block for a subsequent SORTL operation.

Figure 26-8 on page 26-16 illustrates the first and second operands, before and after executing SORTL-SFLR with merge mode equal zero.

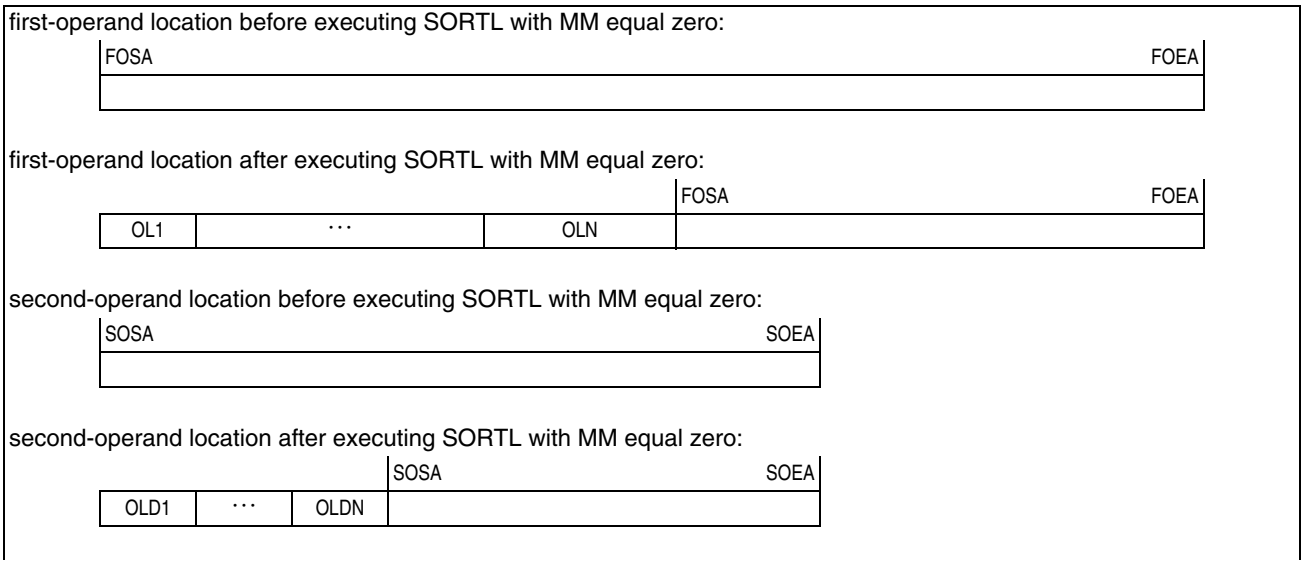


Figure 26-8. First and second operands before and after executing SORTL with MM equal zero (Part 1 of 2)

Explanation:

FOSA	first-operand starting address: location specified by R_1
FOEA	first-operand ending address: location specified by $R_1 + (R_1 + 1) - 1$
OL	output list
OLD	output-list designation
SOSA	second-operand starting address: location specified by R_2
SOEA	second-operand ending address: location specified by $R_2 + (R_2 + 1) - 1$

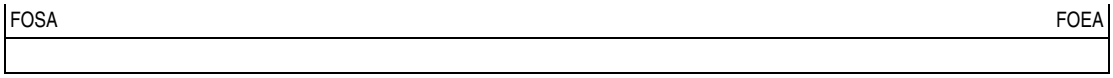
Figure 26-8. First and second operands before and after executing SORTL with MM equal zero (Part 2 of 2)

When merge mode (MM) is one, the active input lists designate lists, each of which is treated as containing records, from left to right, in the sorted order, as specified by the SO field of the parameter block. When MM is one, the records stored to the first-operand location constitute a single output list. When MM is one, each unit of operation consists of the following steps, in the order specified:

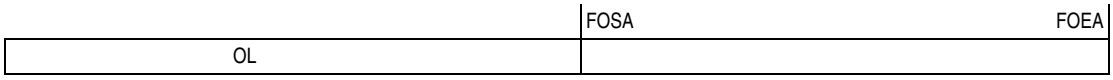
1. Compare the keys of the records designated by all current input-list addresses corresponding to input lists which are active, not empty, and not incomplete. When the sort order is ascending, select the smallest key value and corresponding record. When the sort order is descending, select the largest key value and corresponding record. Refer to page 26-14 for the result of comparing keys of equal value.
2. The selected record is placed at the current first-operand location.
3. The current first-operand address is incremented by the number of bytes equal to the length of the selected record.
4. The current input-list address, corresponding to the input list containing the selected record, is incremented by the number of bytes equal to the length of the selected record.

Figure 26-9 on page 26-17 illustrates the first operand, before and after executing SORTL-SFLR with merge mode equal one.

first-operand location before executing SORTL with MM equal one:



first-operand location after executing SORTL with MM equal one:



Explanation:

FOSA	first-operand starting address: location specified by R_1
FOEA	first-operand ending address: location specified by $R_1 + (R_1 + 1) - 1$
OL	output-list

Figure 26-9. First operand before and after executing SORTL with MM equal one

SORTL-SFLR: Resuming an Operation

When the instruction ends with condition code 1, the program may modify the first-operand address, first-operand length, second-operand address, second-operand length, any active input-list address, and any active-input list length, as appropriate, and subsequently, resume the operation.

When the instruction ends with condition code 2, IILF equal zero, and EILF equal zero, the program may modify the first-operand address, first-operand length, second-operand address, second-operand length, any active input-list address, and any active-input list length, as appropriate, and subsequently, resume the operation.

When the instruction ends with condition code 2 and EILF equal one, the program may modify the input-list address and length for the input list specified by the EILN, as appropriate, and subsequently, resume the operation. In this case, the program may also modify the first-operand address and first-operand length when merge mode (MM) is one.

When the instruction ends with condition code 2 and IILF equal one, the program may modify the input-list address and length for the input list specified by the IILN, as appropriate, and subsequently, resume the operation. In this case, the program may also modify the first-operand address and first-operand length when merge mode (MM) is one.

When the instruction ends with condition code 3, and before reexecuting the instruction to resume the operation, if the program modifies any active input-list address or length, the first-operand address or length, or the second-operand address or length, results are unpredictable.

Figure 26-10 on page 26-18 summarizes the original and final values for inputs to a SORTL-SFLR function that ends in partial completion and which inputs may be modified by the program prior to resuming the operation. It is not required, and is not expected, for the program to modify the parameter block between ending the operation with condition code 3 set and branching back to the instruction, to reexecute the instruction, for the purpose of resuming the operation.

Input to SORTL-SFLR	Before the operation	After partially completing the operation (CC=1, 2, or 3)	Before resuming the operation	After completing the operation (CC=0)
parameter-block-version number	program's choice	unchanged	same as before BO	unchanged
model-version number	not applicable	set	same as after PC	set
sort order	program's choice	unchanged	same as before BO	unchanged
continuation flag	zero required	set to one	same as after PC	set to zero
record-key length	program's choice	unchanged	same as before BO	unchanged
record-payload length	program's choice	unchanged	same as before BO	unchanged
active-input-lists count	program's choice	unchanged	same as before BO	unchanged
empty-input-lists control	program's choice	unchanged	same as before BO	unchanged
empty-input-list flag	not applicable	set	same as after PC	set
empty-input-list number	not applicable	set	same as after PC	set
incomplete-input-list flag	not applicable	set	same as after PC	set
incomplete-input-list number	not applicable	set	same as after PC	set
continuation-record-recall-buffer origin	program's choice	unchanged	same as before BO	unchanged
continuation-state buffer	not applicable	set	same as after PC	undefined
input-listN address	program's choice	modified	restrictions apply - refer to Figure 26-11 on page 26-19	modified
input-listN length	program's choice	modified	restrictions apply - refer to Figure 26-11 on page 26-19	modified
merge mode	program's choice	unchanged	same as before BO	unchanged
first-operand address	program's choice	modified	restrictions apply - refer to Figure 26-11 on page 26-19	modified
first-operand length	program's choice	modified	restrictions apply - refer to Figure 26-11 on page 26-19	modified

Figure 26-10. Summary of Inputs to a SORTL-SFLR Function which may be Resumed (Part 1 of 2)

Input to SORTL-SFLR	Before the operation	After partially completing the operation (CC=1, 2, or 3)	Before resuming the operation	After completing the operation (CC=0)
second-operand address (when MM=0)	program's choice	modified	restrictions apply - refer to Figure 26-11 on page 26-19	modified
second-operand length (when MM=0)	program's choice	modified	restrictions apply - refer to Figure 26-11 on page 26-19	modified
continuation-record-recall buffer (when MM=0)	not applicable	set[1]	same as after PC	undefined

Explanation:

BO	begin operation
MM	Merge Mode
PC	partial completion
[1]	Set in the cases described in the definition of the CRRB origin on page 26-9.

Figure 26-10. Summary of Inputs to a SORTL-SFLR Function which may be Resumed (Part 2 of 2)

Operation ending conditions	output list processing when MM=0	modifications permitted prior to resuming the operation		
		input lists	first operand	second operand
after CC=1	concludes	IL-any: address and length	MM=0: address and length MM=1: address and length	MM=0: address and length MM=1: NA
after CC=2 and EILF=1 (EILN specifies input list which became empty)	continues	IL-EILN: address and length IL-others: none	MM=0: none MM=1: address and length	MM=0: none MM=1: NA
after CC=2 and IILF=0 and EILF=0	concludes	IL-any: address and length	MM=0: address and length MM=1: address and length	MM=0: address and length MM=1: NA
after CC=2 and IILF=1 (IILN specifies the incomplete input list)	continues	IL-IILN: address and length IL-others: none	MM=0: none MM=1: address and length	MM=0: none MM=1: NA
after CC=3	continues	IL-any: none	MM=0: none MM=1: none	MM=0: none MM=1: NA

Explanation:

IL-any	input list with any list number
IL-EILN	input list with list number equal to EILN
IL-IILN	input list with list number equal to IILN
MM	Merge Mode
NA	Not Applicable
concludes	The output list being processed at the end of the operation is not augmented when the operation resumes.
continues	The output list being processed at the end of the operation may be augmented when the operation resumes.
none	Updates not expected: results are unpredictable if specified values are updated under specified conditions.

Figure 26-11. Restrictions for Modifications to Input-List Address and Length fields prior to resuming a SORTL-SFLR Function

SORTL-SFLR: Supplemental Information

Figure 26-12 on page 26-20 illustrates an example of the SORTL-SFLR function with merge mode equal zero. The inputs and resulting outputs are included in the example.

Byte (hex)	input list0	input list1	input list2
0	00...05	00...10	00...99
8	00...01	00...08	00...06
10	00...17	00...02	00...88
18	00...03	00...14	00...20

address (hex)	first operand
1000	00...05
1008	00...10
1010	00...99
1018	00...01
1020	00...06
1028	00...08
1030	00...17
1038	00...88
1040	00...02
1048	00...03
1050	00...14
1058	00...20

address (hex)	second operand
2000	1000
2008	18
2010	1018
2018	28
2020	1040
2028	20

Explanation:

sort order	ascending
record-key length	8 bytes
record-payload length	0 bytes
00...37	8 bytes with most significant 7 bytes being zero and least significant byte containing value 37

Figure 26-12. SORTL-SFLR Example with 3 input lists

When two operations perform the same SORTL-SFLR function with merge mode equal zero on the same set of unsorted input records and the only difference between the two operations is the number of input lists used to specify the input data, the operation with the larger number of input lists typically results in a smaller number of output lists. Figure 26-13 on page 26-21 illustrates an example of using 6 input lists to operate on the same input data as the example in Figure 26-12 on page 26-20, which only uses 3 input lists.

Byte (hex)	input list0	input list1	input list2	input list3	input list4	input list5
0	00...05	00...10	00...99	00...17	00...02	00...88
8	00...01	00...08	00...06	00...03	00...14	00...20

address (hex)	first operand
1000	00...02
1008	00...05
1010	00...10
1018	00...14
1020	00...17
1028	00...88
1030	00...99
1038	00...01
1040	00...03
1048	00...06
1050	00...08
1058	00...20

address (hex)	second operand
2000	1000
2008	38
2010	1038
2018	28

Explanation:

sort order	ascending
record-key length	8 bytes
record-payload length	0 bytes
00...37	8 bytes with most significant 7 bytes being zero and least significant byte containing value 37

Figure 26-13. SORTL-SFLR Example with 6 input lists

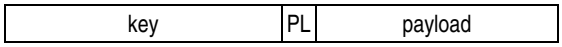
For a suggested procedure to generate a single list of records in sorted order from a set of records in random order, refer to programming note 1 on page 26-24.

Function Code 2: SORTL-SVLR (Sort Variable-Length-Records)

The SORTL-SVLR function operates the same as the SORTL-SFLR function, except for the following:

- All records consist of a fixed-length key, an 8-byte payload length, and a variable-length payload, as shown in Figure 26-14 on page 26-22. Therefore, all records have a variable length.
- Bytes 14-15 of the parameter block for the SORTL-SVLR function are ignored.
- The least significant 2 bytes of the payload-length field of each record contains an unsigned binary integer specifying the length, in bytes, of the payload in the same record. A payload length of zero is valid. The payload length must be a multiple of 8; otherwise a general-operand data exception is recognized. The most significant 6 bytes of the payload-length field are reserved and should contain zeros; otherwise, the program may not operate compatibly in the future. The sum of the key length, eight, and the payload length must not be larger than 4096; otherwise a general-operand data exception is recognized. When a general-operand data exception is recognized as a result of an inappropriate payload length, the input-list address corresponding to the active input list encountering the exception specifies the logical address of the leftmost byte of the errant record. When a variable-length record is stored to the first-operand location, the reserved bytes of the payload-length field are not modified.
- An incomplete input list may not be recognized during a unit of operation which only attempts to reference the key of a record from an input list with an input-list length greater than the key size

and less than the record size. In this case, the incomplete input list will be recognized when attempting to store the record from the incomplete input list, to the first-operand location.



Explanation

PL Payload Length

Figure 26-14. Variable-Length-Record format

The parameter block for the SORTL-SVLR function is the same as the parameter block for the SORTL-SFLR function, except for bytes 14-15, as indicated above.

Refer to section “Other Conditions”, beginning on page 26-22, for descriptions of other conditions that apply to the SORTL-SVLR function.

Special Conditions

A specification exception is recognized when execution of SORT LISTS is attempted and any of the following applies:

- Bits 57-63 of general register 0 designate an unassigned or uninstalled function code.
- The R_1 field designates an odd-numbered register or general register 0.
- The R_2 field designates an odd-numbered register or general register 0. This applies when merge mode (MM) is zero or one.
- The parameter block is not designated on a doubleword boundary.
- The SORTL-SFLR function or the SORTL-SVLR function is specified and the first operand is not designated on a doubleword boundary.
- The SORTL-SFLR or SORTL-SVLR function is specified and the second operand is not designated on a doubleword boundary when MM is zero.

A general-operand data exception is recognized when execution of SORT LISTS is attempted and any of the following applies:

- The SORTL-SFLR or SORT-SVLR function is specified and no bits, or multiple bits, of bits 0-7 of the parameter-block-version number, contain a value of one.
- The SORTL-SFLR or SORTL-SVLR function is specified and the size or format of the parameter block, as specified by the parameter-block-version number, is not supported by the model.
- The SORTL-SFLR or SORTL-SVLR function is specified and the record-key length specifies a key size of zero, a key size which is not a multiple of 8, or a key size greater than 4096.
- The SORTL-SFLR function is specified and the record-payload length in the parameter block specifies a payload size which is not a multiple of 8, or a payload size, when added to the key size, is greater than 4096.
- The SORTL-SVLR function is specified and the record-payload length in the variable-length record specifies a payload size which is not a multiple of 8, or a payload size, when added to the key size, is greater than 4088, in which case it is model dependent whether the operation is suppressed or terminated.
- The SORTL-SFLR or SORTL-SVLR function is specified and the value of the active-input-lists count code (AILCC) plus one is greater than the number of input lists described by the parameter block.
- The SORTL-SFLR or SORTL-SVLR function is specified and an input-list address, corresponding to an active input list, is not designated on a doubleword boundary.

When a general-operand data exception is recognized, the operation is suppressed, except as described above.

Other Conditions

Execution of the instruction ends after processing a CPU-determined number of bytes. This permits interruptions to occur. When the instruction ends with partial completion, the addresses in general registers R_1 and R_2, the lengths in general registers $R_1 + 1$ and $R_2 + 1$, and specific fields of the parameter block are

updated so that the instruction, when reexecuted, resumes the operation at the point it was suspended.

Access exceptions are not recognized for locations greater than 4 K-bytes to the right of the location designated by the first-operand address. Access exceptions are not recognized for locations greater than 4 K-bytes to the right of the location designated by an input-list address.

If an access exception is due to be recognized for the first operand, second operand, or any active input list, the result is that either the exception is recognized or condition code 3 is set. If condition code 3 is set, the exception will be recognized when the instruction is executed again to continue processing the same operands, assuming the exception condition still exists.

Access-exception conditions are not recognized for an inactive input list. Access-exception conditions are not recognized for an input list with an input-list length equal to zero at the beginning of the execution of the instruction.

When the key of a record crosses a page boundary and access-exception conditions exist for both pages, either access exception may be recognized.

When access-exception conditions exist for multiple keys being processed during a single unit of operation, any of these conditions may be recognized.

When the parameter block crosses a page boundary and access-exception conditions exist for both pages, the access exception for the leftmost page is recognized. Access exceptions apply to the entire parameter block, regardless of any input list being in the inactive state.

When the operation ends with partial completion, up to 4 K-bytes of data may have been stored at locations within the first operand which are at, or to the right of, the location designated by the updated first-operand address. Such stores result in setting change bits, when applicable, and recognizing PER storage-alteration events, when applicable. Storing to these locations will be repeated when the instruction is executed again to continue processing the same operands.

As observed by this CPU, other CPUs, and channel programs, references to the parameter block, the first operand, the output-lists-delineations buffer, and all input lists in the active state may be multiple-access references, accesses to these storage locations are not necessarily block-concurrent, and the sequence of these accesses or references is undefined.

Results are unpredictable when the specified function is SORTL-SFLR or SORTL-SVLR and any of the following apply:

- The parameter block overlaps any active input list or the first operand.
- Any active input list overlaps the first operand.
- Merge mode is zero and the parameter block overlaps the second operand or the continuation-record-recall buffer.
- Merge mode is zero and any active input list overlaps the second operand or the continuation-record-recall buffer.
- Merge mode is zero and the first operand overlaps the second operand or the continuation-record-recall buffer.
- Merge mode is zero and the second operand overlaps the continuation-record-recall buffer.
- Another CPU or channel program stores to a key of a record in an input list or the continuation-record-recall buffer.

Resulting Condition Code:

0 Normal completion
1 The length of the first operand is less than the size of a record, or merge mode is zero and the length of the second operand is less than 16
2 An incomplete input list was encountered (IILF=1), or the EILCL is nonzero and the length of an input list became equal to zero during the operation
3 CPU-determined amount of data processed

Program Exceptions:

- Access (fetch, input lists; fetch and store, parameter block and continuation-record-recall buffer; store, operands 1 and 2)
- Data with DXC 0, general operand
- Operation (if the enhanced-sort facility is not installed)
- Specification
- Transaction constraint

The priority of execution for the SORT LISTS instruction is shown in Figure 26-15 on page 26-24. When multiple conditions which have priority values beginning with 13 exist, the condition recognized is the one that is encountered first, as the operation proceeds. When the operation is being resumed (the continuation flag is one at the beginning of the execution of the instruction), a history of prior comparisons between keys may be used in place of initially accessing all input lists which are active and not empty. As a result, an access exception for an access to a specific input list may not be encountered at the same point of processing, as compared to when no history of prior comparisons is used. When variable-length records are processed, conditions which are a function of a record length may be partially evaluated before the payload length is determined, and completely evaluated after the payload length is determined. As a result, the observed priority among such conditions may differ when a condition is determined to exist after only partially evaluating requirements, instead of after completely evaluating all requirements.

1.-6.	Exceptions with the same priority as the priority of program-interruption conditions for the general case.
7.A	Access exceptions for second instruction halfword.
7.B	Operation exception.
7.C	Transaction constraint.
8.A	Specification exception due to invalid function code or invalid register number.
8.B	Specification exception due to parameter block not designated on doubleword boundary.
8.C	Specification exception due to first operand not designated on doubleword boundary.
8.D	Specification exception due to second operand not designated on doubleword boundary and merge mode is zero.
9.	Access exceptions for an access to bytes 0-7 of the parameter block.
10.	General-operand data exception due to an unsupported value of the PBVN field in the parameter block.
11.	Access exceptions for an access to bytes of the parameter block other than bytes 0-7.

Figure 26-15. Priority of Execution: SORTL (Part 1 of 2)

12.	General-operand data exception due to an invalid value of a field in the parameter block other than the PBVN.
13.A	Access exceptions for an access to an active input list.
13.B	Access exceptions for an access to the continuation-record-recall buffer when merge mode is zero.
13.C	Access exceptions for an access to the first operand.
13.D	Access exceptions for an access to the second operand when merge mode is zero.
13.E	Condition code 2 due to an incomplete input list.
13.F	Condition code 1 due to insufficient length of first operand.
13.G	Condition code 1 due to insufficient length of second operand when merge mode is zero.
13.H	General-operand data exception due to an invalid payload length of a variable-length record.
13.I	Condition code 2 due to an empty input list.
14.	Condition code 3.

Figure 26-15. Priority of Execution: SORTL (Part 2 of 2)

Programming Notes:

1. To generate a single list of records in sorted order from a set of records in random order, a program may perform the following procedure:

   a. Evenly partition the set of records among an initial set of lists, where each list contains records in random order. Execute the SORT LISTS instruction with the initial set of lists as input lists and merge mode equal to zero, to generate an intermediate set of lists (each of which contains records in sorted order), and the storage locations and lengths for each list of the intermediate set of lists.

   b. Execute the SORT LISTS instruction with the intermediate set of lists as input lists and merge mode equal to one, to generate the final and single list, which contains all records in sorted order.

2. The intended values and uses of the empty-input-lists control (EILCL) field are as follows:

EILCL (binary)	description
00	Stop after all records from all active input lists are sorted.
10	Stop after input list0 (always active) becomes empty.
11	Stop after any active input list becomes empty. When the operation ends for this reason, EILF is set to one and EILN identifies the input list which became empty.

3. When the active-input-lists count code (AILCC) is zero, there is only one active input list and the results stored at the first-operand location are the same as the data fetched from input list0.

4. The total execution time of the instruction is generally proportional to the size of the keys. Larger keys generally result in longer execution times. Total execution time of the instruction is also proportional to the size of the payloads.

5. Future models may only support a maximum number of input lists which is less than the maximum number supported by models currently available. The interface size of 32 input lists is always available on all models.

6. Models implementing separate instruction and data caches may use the instruction cache to perform storage-operand fetch references to data in active-input lists. When a program performs storage-operand store references to storage locations, and in a relatively-short period of time, executes SORT LISTS with those same storage locations being designated as active input lists, performance of the program may be affected due to transferring the data associated with those storage locations from the data cache to the instruction cache.

7. Special precautions should be taken if SORT LISTS is made the target of an execute-type instruction. See the programming note concerning interruptible instructions under EXECUTE.

8. When a program expects to invoke SORT LISTS multiple times with merge mode equal to zero, as part of processing a large amount of data, it is recommended the program utilize all input lists available and evenly partition records among the input lists. This reduces the number of times the data is accessed when sorting all of the data.

9. Subsequent to SORT LISTS with merge mode equal to zero ending with condition code 0 set and multiple output-list delineations (OLDs) in the second operand, a program intending to generate a single list of records in sorted order must invoke another SORT LISTS operation with input lists specified to be the resulting OLDs from the prior SORT LISTS invocation. In this case, it is recommended the second invocation of SORT LISTS specifies merge mode equal to one.

   Similarly, subsequent to invoking SORT LISTS with merge mode equal zero, for as many times as necessary, to generate a complete set of sorted lists from a large number of randomly ordered records, it is recommended to invoke SORT LISTS with merge mode equal one, for as many times as necessary, to generate a single sorted list.

10. To reduce the number of times each record is accessed when merging multiple sorted lists into a single list with ascending sort order (for example), it is recommended the program performs the following process:

    - Determine the maximum number, N, of input lists available for SORT LISTS.
    - Compare the keys of the first record of all sorted lists which have not yet been merged into the single list. Select the N lists which have the lowest first key values.
    - Execute SORT LISTS with merge mode (MM) equal one, empty-input-lists control (EILCL) equal 10 binary, input-list0 specifying only the first record of the list with the highest first key value of the selected N lists, and the remaining input lists specifying the other N-1 selected lists.
    - Subsequent to SORT LISTS ending with condition code 2, IILF equal zero, and EILF equal zero, repeat the process.

11. Subsequent to SORT LISTS ending with condition code 1 set, it is recommended the program perform the following actions prior to invoking SORT LISTS again, to resume the operation:

    - If the first-operand length is less than the largest record length of all records being processed, then the first-operand length or first-operand address and length should be updated, as appropriate.

- If merge mode (MM) is zero and the second-operand length is less than 16, then the second-operand length or second-operand address and length should be updated, as appropriate.
- If the length of any active input list equals zero, then the corresponding input-list address and length may be updated to designate another list of records to be included in the sorting operation.

12. Subsequent to SORT LISTS ending with condition code 2 set, it is recommended the program perform the following actions prior to invoking SORT LISTS again, to resume the operation:

    - If the incomplete-input-list flag (IILF) is one, then the input-list length or input-list address and length of the input list identified by the incomplete-input-list number (IILN) should be updated, as appropriate.
    - If the empty-input-list flag (EILF) is one, then the input-list length or input-list address and length of the input list identified by the empty-input-list number (EILN) should be updated, as appropriate.
    - If the IILF is zero, the EILF is zero, and the input-list0 length is zero, then the input-list0 length or input-list0 address and length should be updated, as appropriate. Furthermore, the input-list address and length for all active input lists may be updated, which may be the appropriate action if there was only one record designated by input-list0 originally, and the empty-input-lists control (EILCL) is 10 binary.
    - If merge mode (MM) is one and the first-operand length is less than the largest record length of all records being processed, then the first-operand length or first-operand address and length should be updated, as appropriate.
    - If MM is zero and either IILF is one, or EILF is one, then the first-operand address and length, and the second-operand address and length should not be updated.
    - If MM is zero, IILF is zero, EILF is zero, and the first-operand length is less than the largest record length of all records being processed, then the first-operand length or first-operand address and length should be updated, as appropriate.
    - If MM is zero, IILF is zero, EILF is zero, and the second-operand length is less than 16, then the second-operand length or second-operand address and length should be updated, as appropriate.

13. When the next-sequential instruction after NEXT INSTRUCTION ACCESS INTENT (NIAI) is SORT LISTS (SORTL), the execution of SORTL is not effected by NIAI.

⋮

End of Document

SA22-1082-00